Proceedings Paper:
Calinescu, Radu Constantin orcid.org/0000-0002-2678-9260 and Yonbawi, Saud Reda (2018) Towards a Hierarchical-Control Architecture for Distributed Autonomous Systems. In: Workshop on Verification and Validation of Autonomous Systems. .

Towards a Hierarchical-Control Architecture for Distributed Autonomous Systems

Radu Calinescu and Saud Yonbawi
Department of Computer Science, University of York, UK

1 Introduction and Architecture Overview

Recent technological advances enable the cost-effective manufacturing of a wide range of distributed autonomous systems. However, deploying these systems in safety-critical domains (e.g., transportation and manufacturing) is very challenging, as they need to comply with strict functional and nonfunctional requirements in uncertain operating environments [4, 7]. System-level formal verification cannot be used to ensure this compliance because it does not scale to systems of this size and complexity, especially if the verification needs to be repeated at runtime, after each change that affects one of the system components.

To address this limitation, we propose a hierarchical-control architecture based on two-level formal verification. The new architecture comprises (i) a decentralised system-level control loop that partitions the goals of the distributed autonomous system among its components *under conservative assumptions*, and (ii) component-level control loops responsible for achieving the component sub-goals. The operation of the two control loops is underpinned by formal models that are small enough to enable their verification at runtime.

As shown in Fig. 1, an instance of the system-level monitor-analyse-plan-execute (MAPE) control loop is running on each system component. Its Local Monitor oversees the operation of the component-level MAPE loop, detecting difficulties with the achievement of the component-level goals, while the Peer Monitor receives notifications of such difficulties experienced by other components. The Analysis uses formal verification to compute alternative contributions that the local component can make to the achievement of the system goals, and the cost(s) of each of these contributions. For example, for a robotic team used in a search and rescue mission, each robot may be able to cover different parts of the search area, with different (predicted) energy consumption, risks, etc. This *local capability analysis* is carried out infrequently, i.e., when the component joins the system, and when the Local Monitor notices that the component-level planner P experiences difficulties.

The Planning uses formal techniques to partition the system goals into component (sub)goals based on the alternative component-level contributions "proposed" by the locally running Analysis and by the Analysis instances running on the other components, where the latter information is obtained as a summary, via the Peer Monitor. This goal partition is carried out such as to guarantee that the system goals are achieved, e.g. by allocating each part of the search and rescue area to at least one robot. The Execution configures the component-level MAPE in line with the goals that the Planning allocated to the local component.

A key benefit of our solution is the use of *conservative assumptions* in the Analysis step of the system-level MAPE loop. For instance, the Analysis assumes that a search-and-rescue robot may move (slightly) slower, consume more energy and encounter more obstacles than in its past missions or mission simulations.

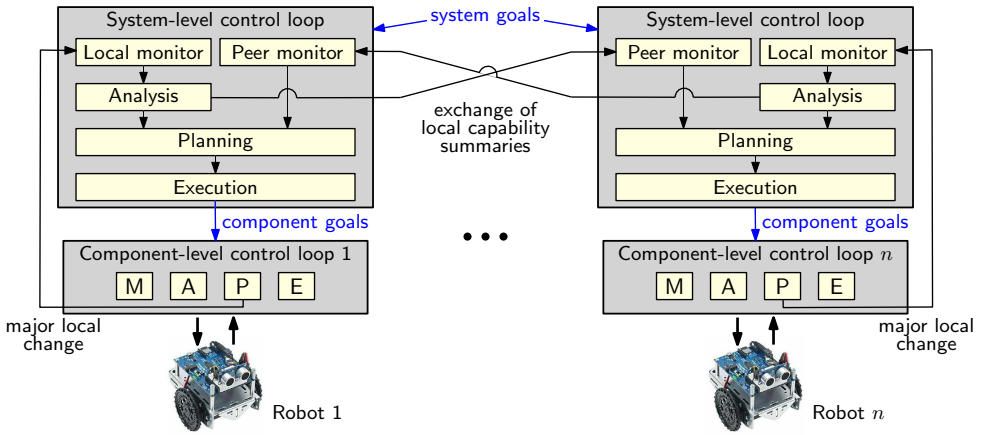

Figure 1: Hierarchical-control architecture applied to search-and-rescue robotic team

The "slack" provided by these conservative assumptions allows the component-level MAPE loops to handle environment uncertainties and changes that do not invalidate the assumptions. Using different slack levels enables a wide range of trade-offs between the efficiency with which the system components operate, and the frequency with which the component-level MAPE loops[1] become unable to synthesise a component-level plan after environmental changes and need to report a "major local change" to the Local Monitor. Thus, a robot that assumes an up to 20% higher than normal energy use will be less likely to encounter situations where this assumption is violated than a robot that assumes only up to 10% extra energy use; but the former robot would be allocated a smaller part of the search area in the Planning step than the latter robot.

2 Supported Verification Techniques and Preliminary Evaluation

Our approach is not prescriptive about the verification techniques used by its two MAPE loops. However, the Analysis step of the system-level MAPE loop and the analysis and planning steps of the component-level MAPE loop must employ techniques based on the same modelling paradigm. This is needed because the Analysis needs to predict (for conservative ranges of environmental parameter values) what the component-level MAPE loop will analyse and plan for the actual values of these parameters. As in recent work on developing self-adaptive systems with strict requirements, stochastic models ranging from Markov chains to Markov decision processes and stochastic multiplayer games [3] are particularly suitable for modelling the environmental uncertainty of autonomous systems—with probabilistic [2] or statistical [6] model checking used to analyse and synthesise [5] these models within the two MAPE loops.

To capture this generality of the architecture, we prototyped it as a reusable set of abstract Java classes, and we specialised these classes for the control of a team of three Parallax ActivityBot mobile robots involved in a lab-based search and rescue application around a predefined perimeter. The application (not described here due to limited space) uses continuous-time Markov chains to model the individual robot behaviour locally, and probabilistic model checking to predict energy use for alternative search areas, with the system-level goal of continuing the mission for as long as possible given the available battery energy of each robot.

3 Related Work

The hierarchical-control architecture from Fig. 1 extends our recent work on decentralising the control loops of self-adaptive systems [1] and generalises its system-level loop using different patterns similar to [8]. There is a significant body of work on architectures that could support self-adaptation in distributed autonomous systems, although most approaches that we are aware of focus on specific application domains, use fixed modelling and analysis techniques, or cannot provide the same levels of assurances as our approach. Due to limited space, we cannot detail these alternative approaches. However, we want to emphasise that a general-purpose architecture based on formal verification (like the one we proposed) is essential for lowering the engineering effort required to develop trustworthy distributed autonomous systems for safety-critical domains [2, 4, 7].

References

[1] R. Calinescu et al. Self-adaptive software with decentralised control loops. In *Fundamental Approaches to Software Engineering (FASE)*, pages 235–251, 2015.

[2] R. Calinescu et al. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Trans. Software Eng.*, PP(99):1–31, 2017.

[3] R. Calinescu et al. Synthesis and verification of self-aware computing systems. In *Self-Aware Computing Systems*, pages 337–373. Springer, 2017.

[4] R. de Lemos et al. Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 3–30. Springer, 2017.

[5] S. Gerasimou et al. Search-based synthesis of probabilistic models for quality-of-service software engineering. In *ASE'15*, pages 319–330, 2015.

[6] M. U. Iftikhar and D. Weyns. Towards runtime statistical model checking for self-adaptive systems. Technical Report CW 693, KU Leuven, 2016.

[7] Lloyd's Register Foundation. Foresight review of robotics and autonomous systems, 2016.

[8] D. Weyns et al. On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, pages 76–107, 2010.

[1] We assume that the component-level MAPE loops are implemented using established approaches (e.g., [2]), so we are not describing them in the paper.